# LArFlow: From 2D images to 3D space-points

Taritree Wongjirad (Tufts)

Exa.TrkX Workshop

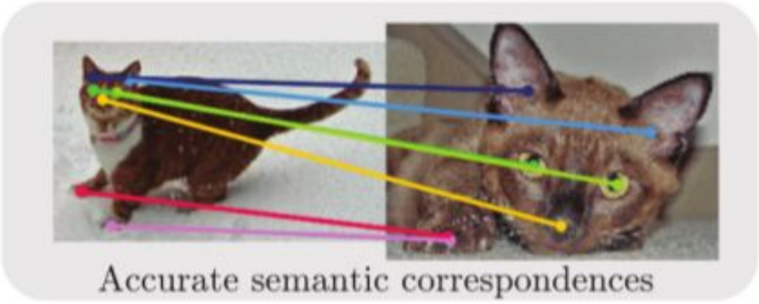
June 4th, 2019

# Introduction

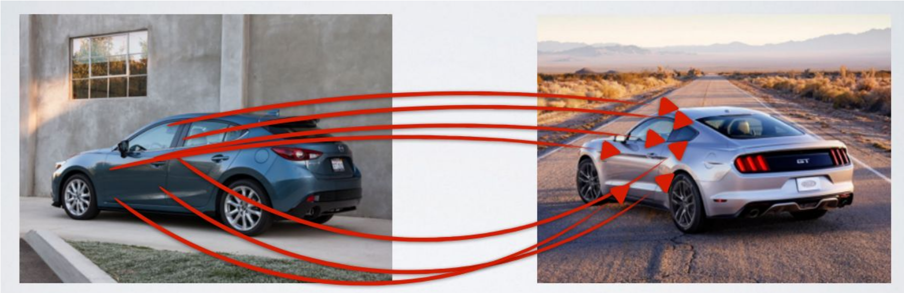
- Developing convolutional neural network (CNN) to generate 3D point cloud from 2D LArTPC images
- This work follows the computer vision efforts in dense correspondence
- Discuss
  - Network architecture
  - Data preparation
  - Post-processing
  - Preliminary performance metrics
- Next steps

# Dense pixel correspondence

- Goal of dense pixel correspondence in the computer vision world -- match regions of one image to another, connecting semantically similar regions

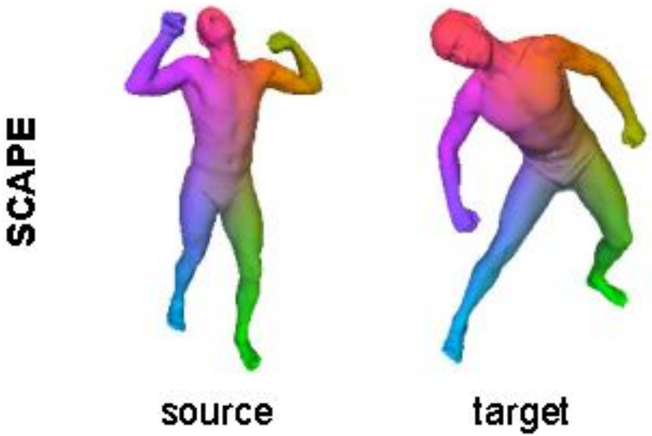



Choy et al. “Universal Correspondence Network” NIPS 2016



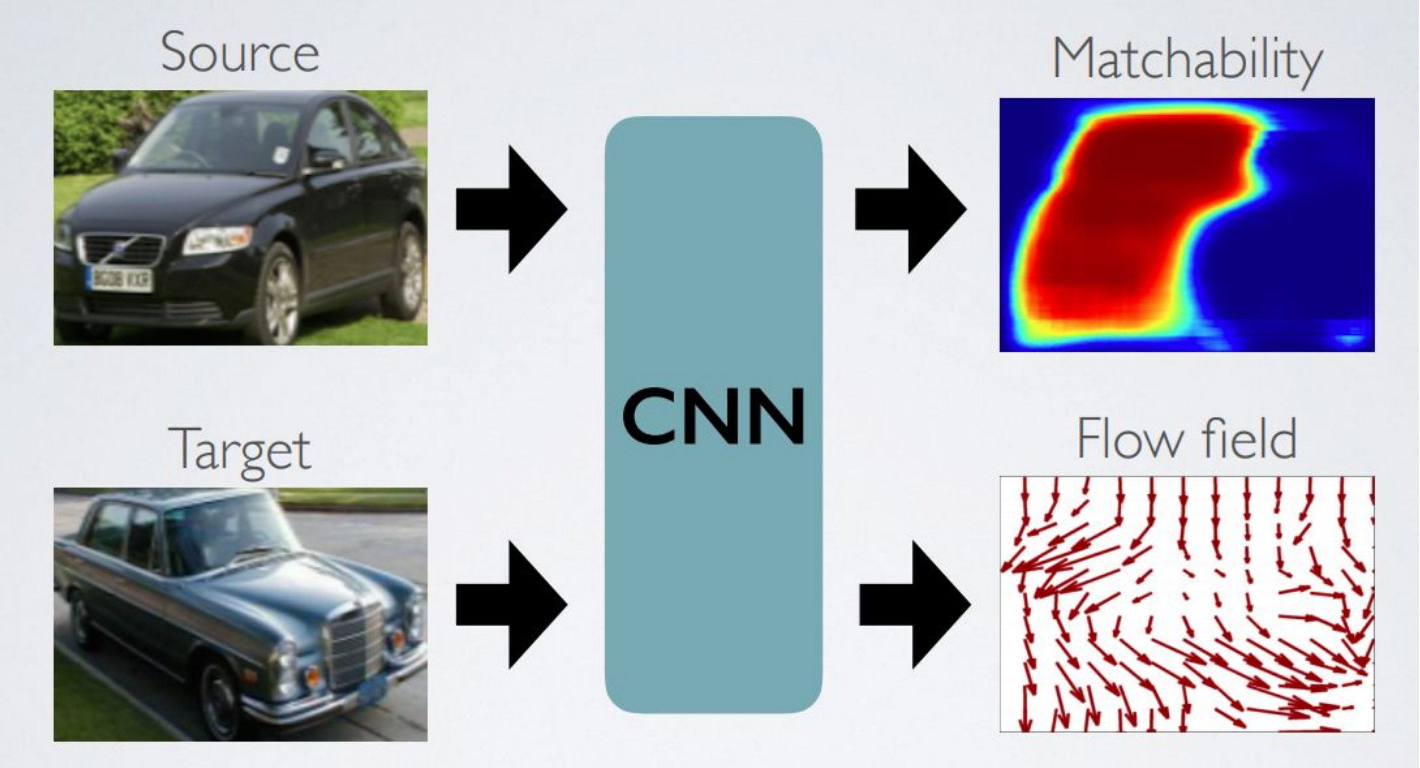
Zhou, Krähenbühl et al. “Learning Dense Correspondence via 3D-guided Cycle Consistency” CPVR 2016

colors indicate what should be matched




Wei et al. “Dense Human Body Correspondences Using Convolutional Networks” CPVR 2016

# Dense Pixel Correspondence: Example output




<u>in LArTPC context</u>

matchability = 0 when true target pixel in dead wires, below thresh, etc.

enforce same-time tick, so only wire-direction flow predicted

*We use 512 time bin x 512 wires cropped images for training due to technical restraints

# LArTPC pixel correspondence: LArFlow

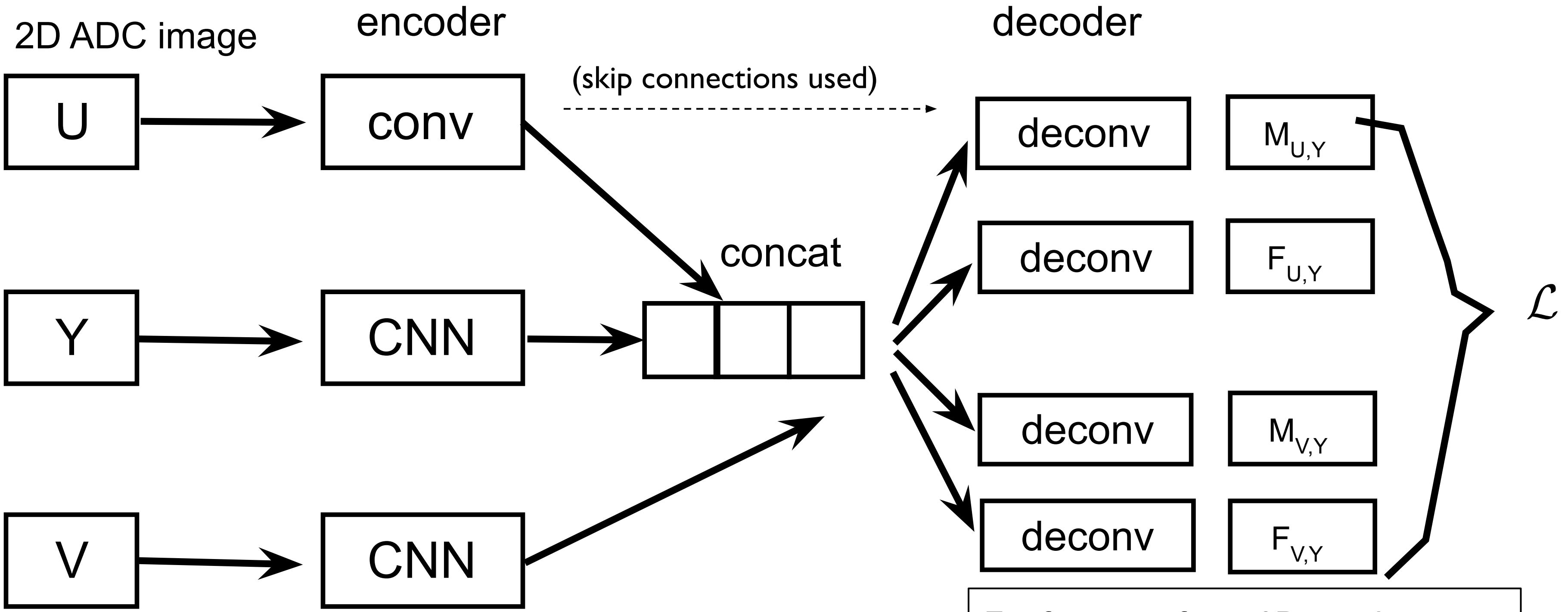
Network predicts correspondence between pixels (charges) in Y, U, V ADC images

For pixel i in Y plane: the CNN is asked to predict ***shift*** needed to move to pixel 1175

Which is where the corresponding pixel is in U plane


Correspondence prediction gives 3D space-point for that charge

# LArFlow network architecture

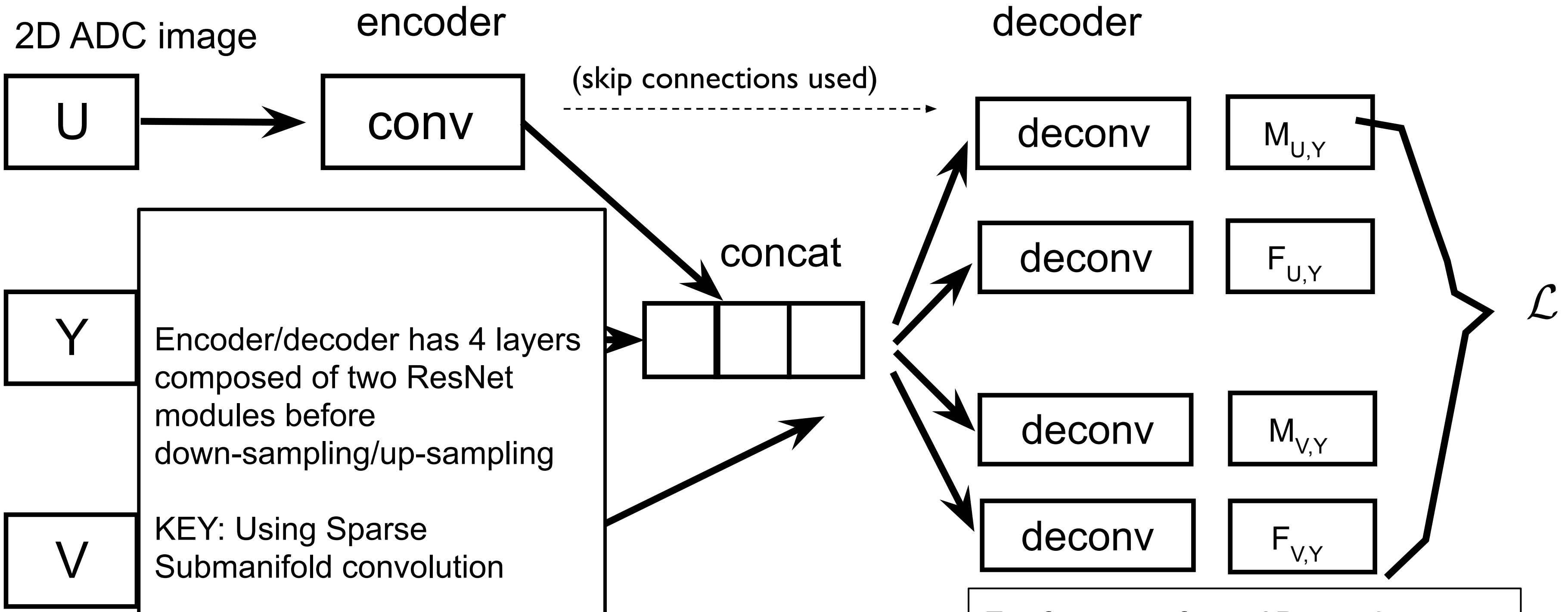



Loss function

$$\mathcal{L} = \lambda M + F$$

For future: enforce 3D consistency loss between Y->U and Y->V prediction

# LArFlow network architecture




Encoder/decoder has 4 layers composed of two ResNet modules before down-sampling/up-sampling

KEY: Using Sparse Submanifold convolution

Loss function

$$\mathcal{L} = \lambda M + F$$

For future: enforce 3D consistency loss between Y->U and Y->V prediction

# Loss

- Smooth L1 loss used (regression loss)
  - Error between true flow and predicted flow
  - Not capping pixels with large differences (sometimes done in the literature to prevent influence of outlier)
- Only calculate loss on pixels where at least one end of flow has charge
  - Allowing predictions from dead regions in starting image to charge on target region and vice-versa
  - These harder pixels allowed as first results showing network was getting these cases fairly well even though such cases were not included in calculated loss

# Data preparation

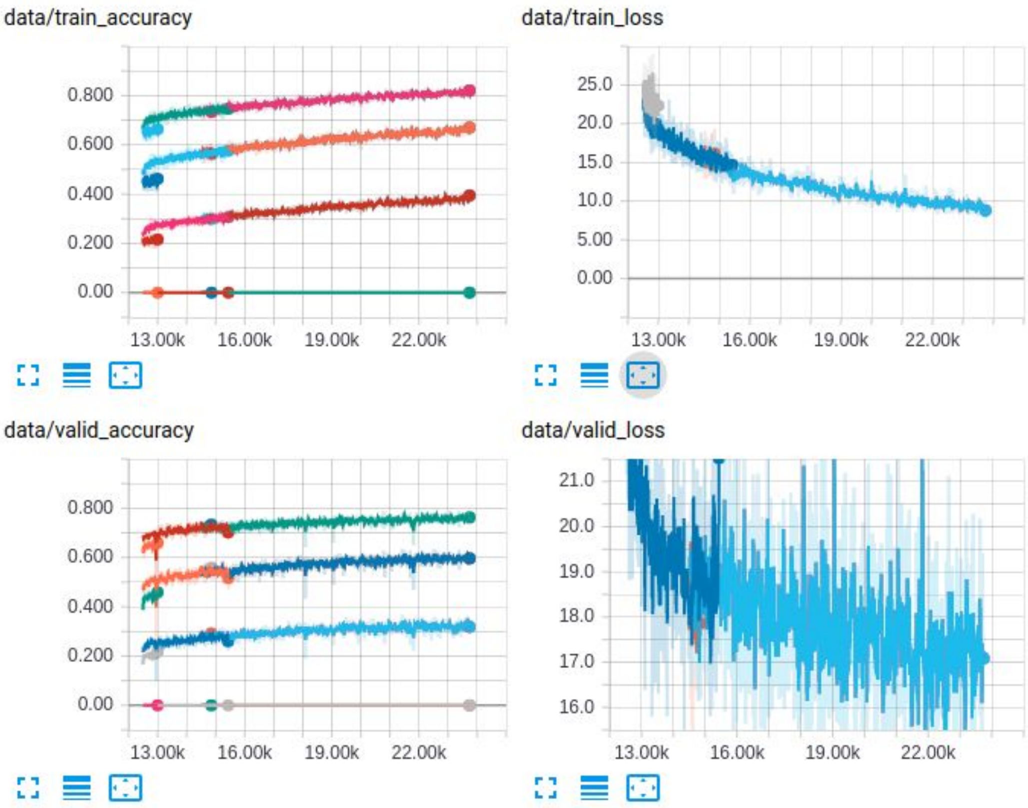
- Images preparation:
  - Noise filtering
  - pulse finding + zero suppression
  - Deconvolve wire response
    - Accounting for electronics response + expected induced signal
  - Downsample in time (summed) by factor of 6
- 3D consistent cropping
  - Full size: 3456 (wire) x 6448 (ticks)
  - Downsampled size: 3456 x 1008 -- both dimensions about 3 mm
  - Cropped into 832 wire x 512 ticks (24 images per plane)

# Data preparation

- Images preparation:
  - Noise filtering
  - pulse finding + zero suppression
  - Deconvolve wire response
    - Accounting for electronics response + expected induced signal
  - Downsample in time (summed) by factor of 6
- 3D consistent cropping
  - Full size: 3456 (wire) x 6448 (ticks)
  - Downsampled size: 3456 x 1008 -- both dimensions about 3 mm
  - Cropped into 832 wire x 512 ticks (24 images per plane)
- 220k training images, 40k validation images
  - Simulated images -- truth used to produce labels

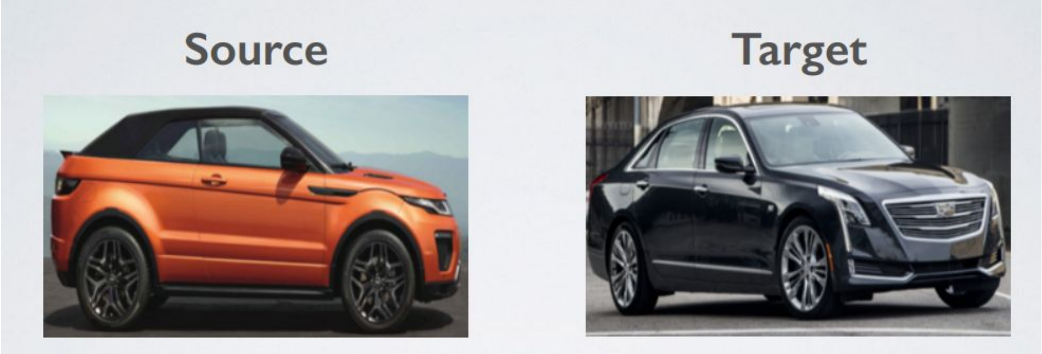
# Training

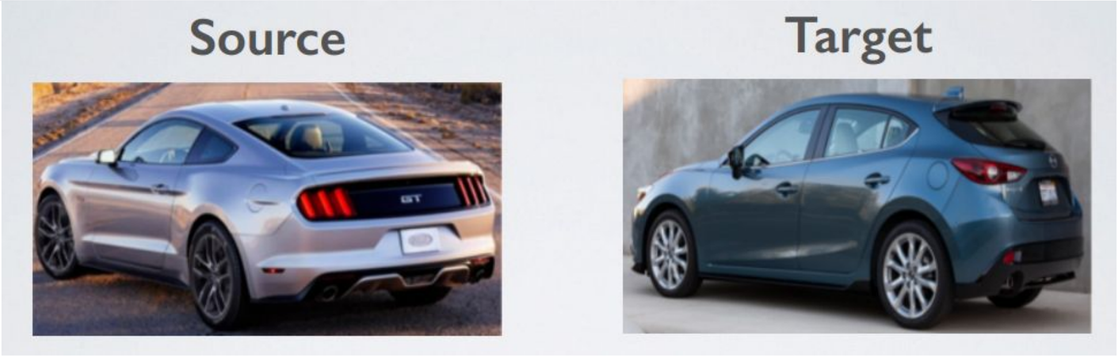
- First passes
- Training network to project charge on Y wire to charge on U wire
- Accuray Curves measure fraction where |Predicted -true| projected U plane pixel is
  - <10 pixels
  - <5 pixels
  - <2 pixels
- For validation sample
  - <10: 78%
  - <5: 60%
  - <2: 32%



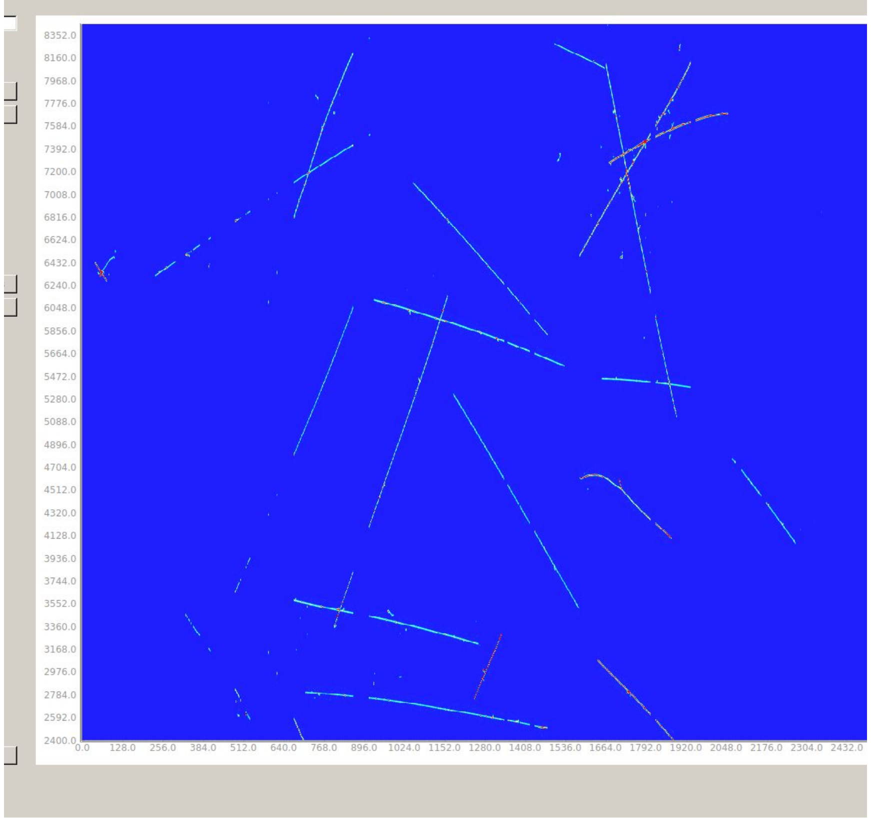
# Visualization

- Visualize how the network is doing by projecting source image into target image (masking out matchibily=0 pixels)

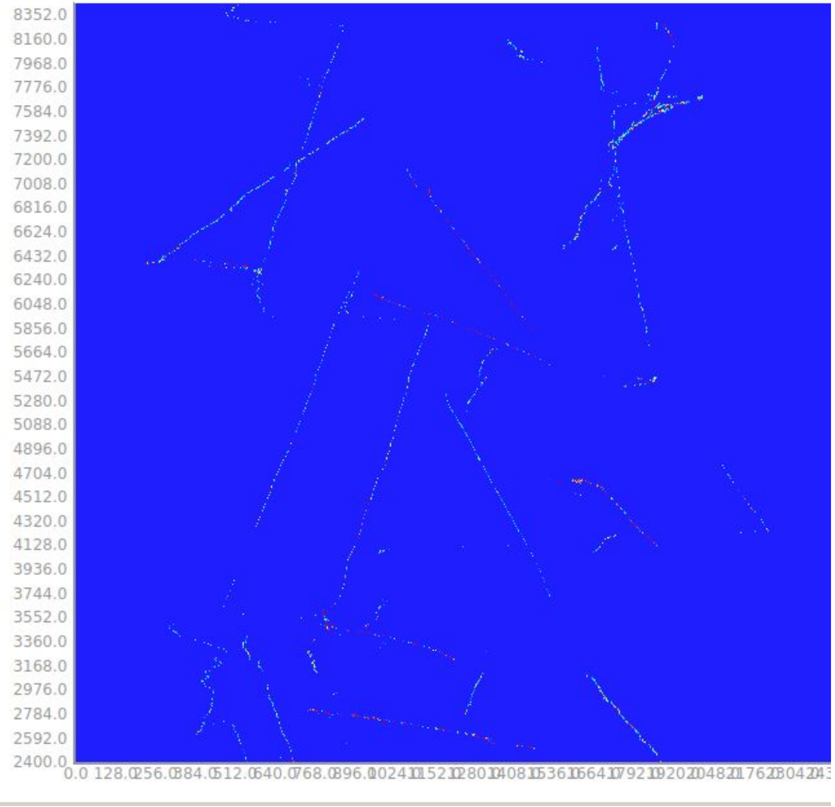



# Visualization



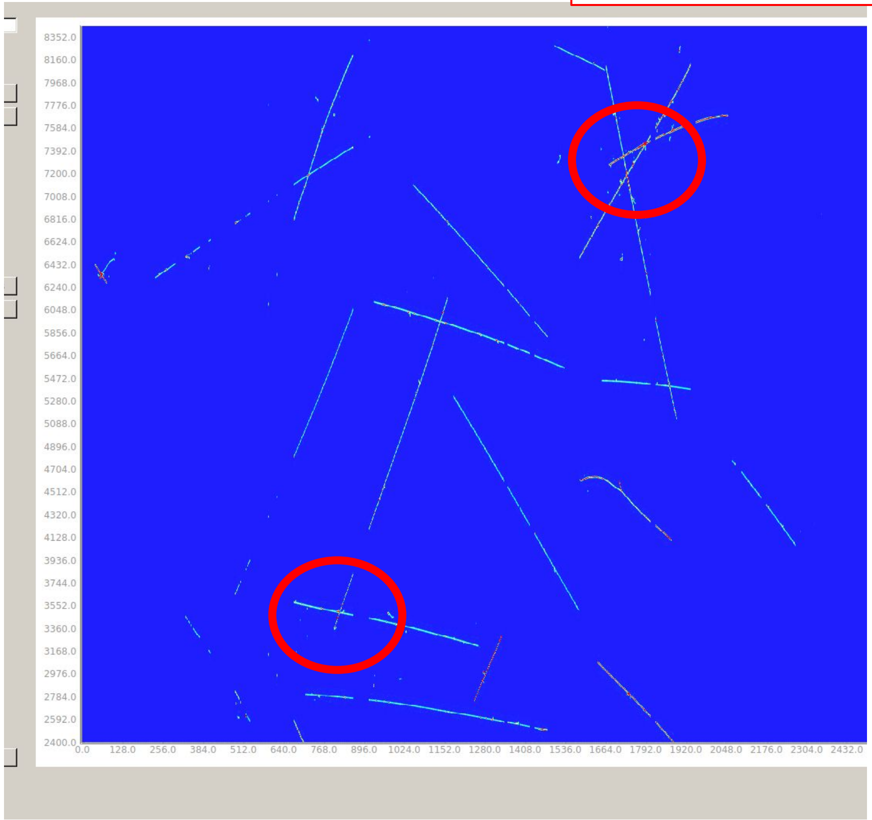

What the U-plane ADC is



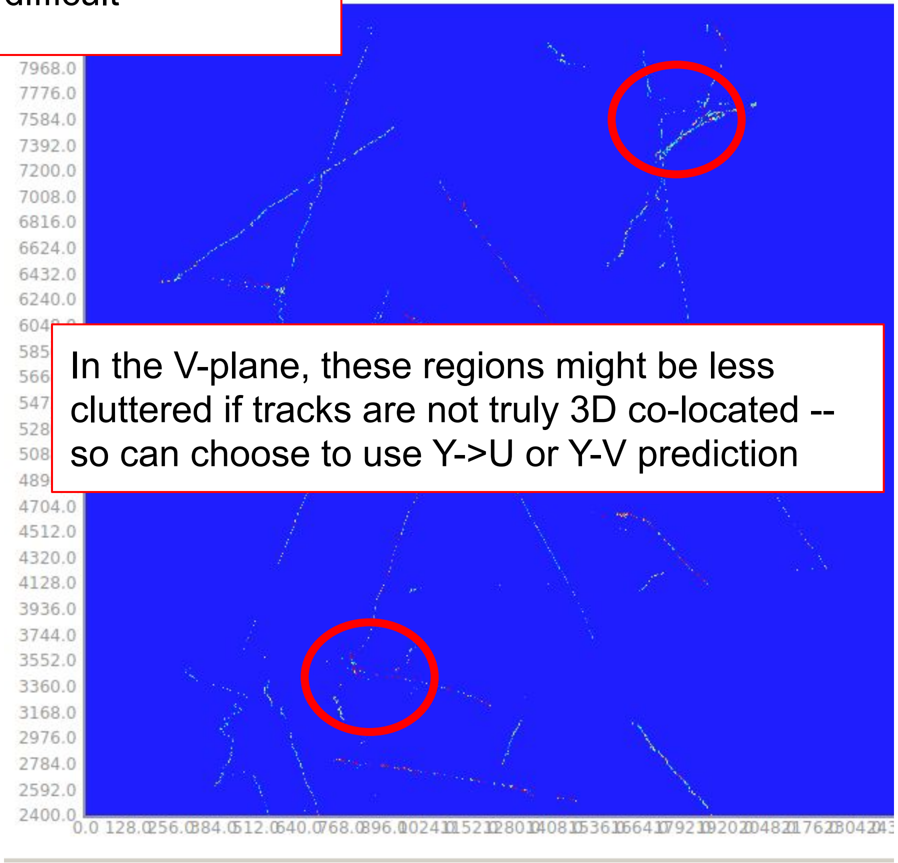

Taking y-plane charge, and moving it to predicted u-plane location

Visualization

Getting many mappings mostly right
But busy regions are difficult




What the U-plane ADC is

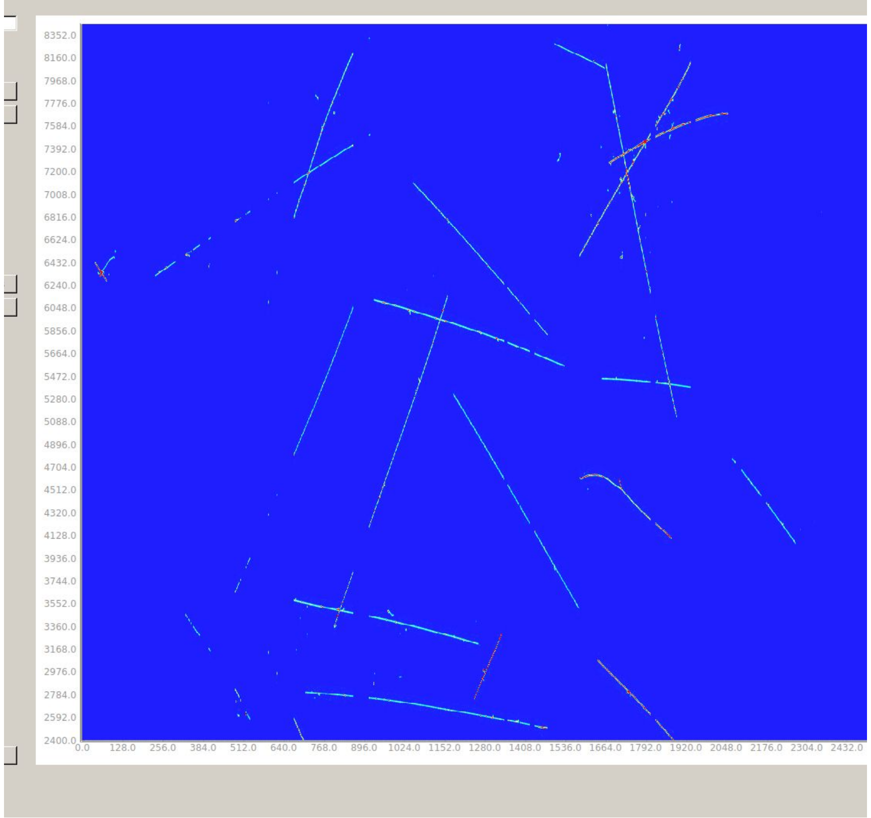



In the V-plane, these regions might be less cluttered if tracks are not truly 3D co-located -- so can choose to use Y->U or Y-V prediction

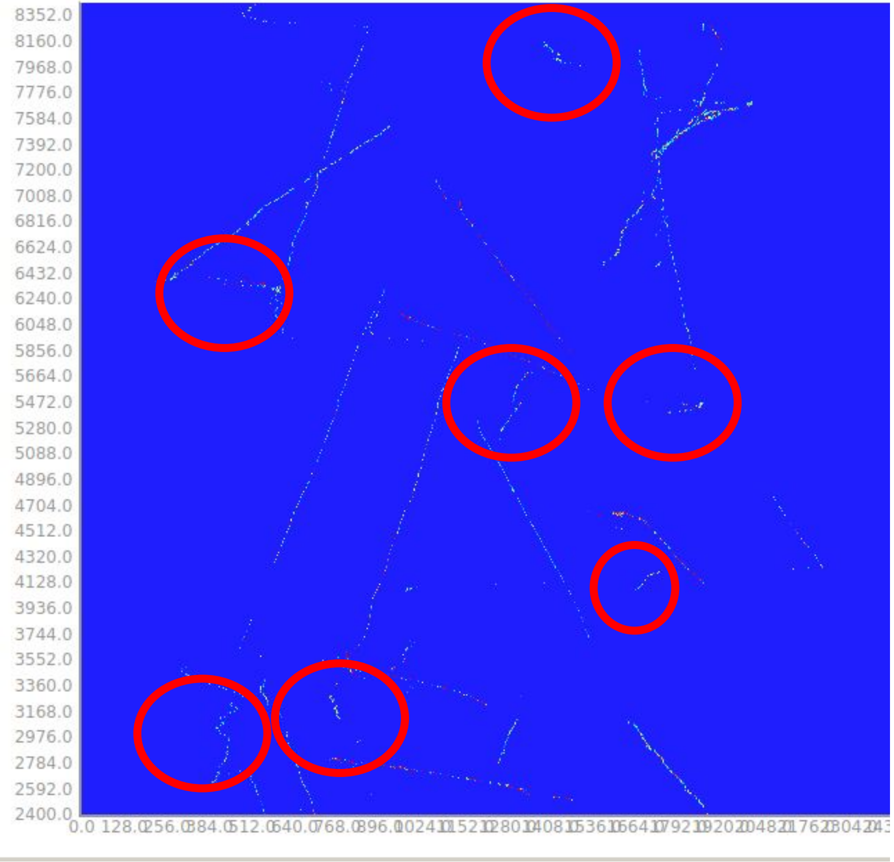
Taking y-plane charge, and moving it to predicted u-plane location

Visualization

More errors (investigating why)



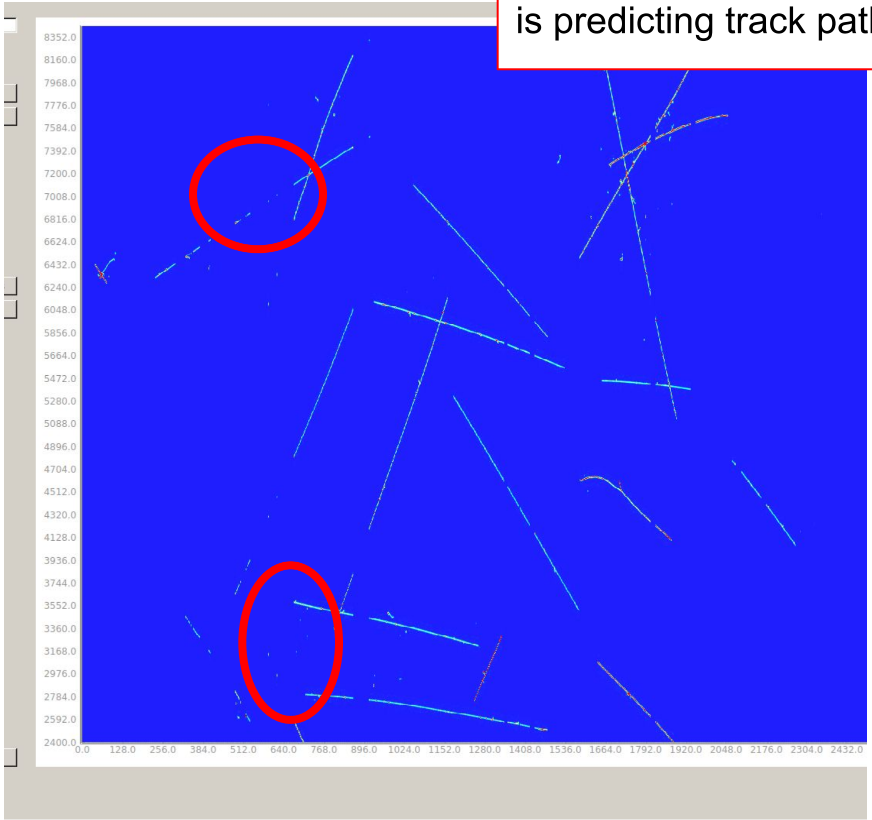

What the U-plane ADC is



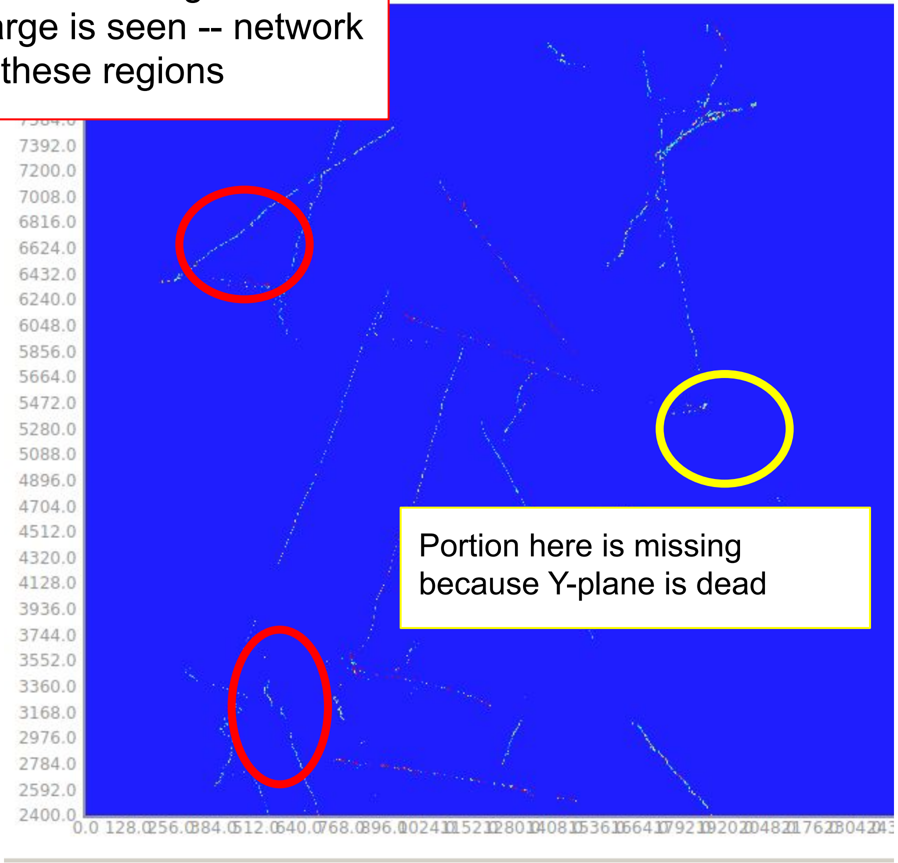

Taking y-plane charge, and moving it to predicted u-plane location

# Visualization

Note: projecting y-pixels into dead regions in the U-plane where no charge is seen -- network is predicting track path in these regions

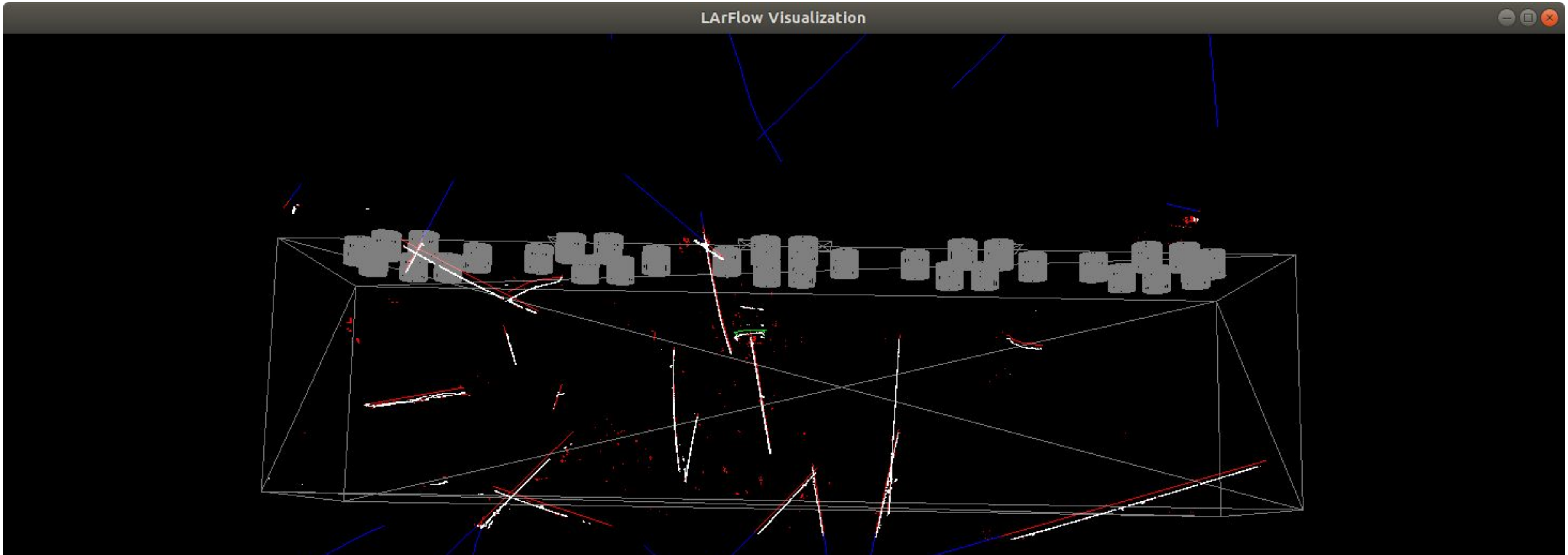


What the U-plane ADC is



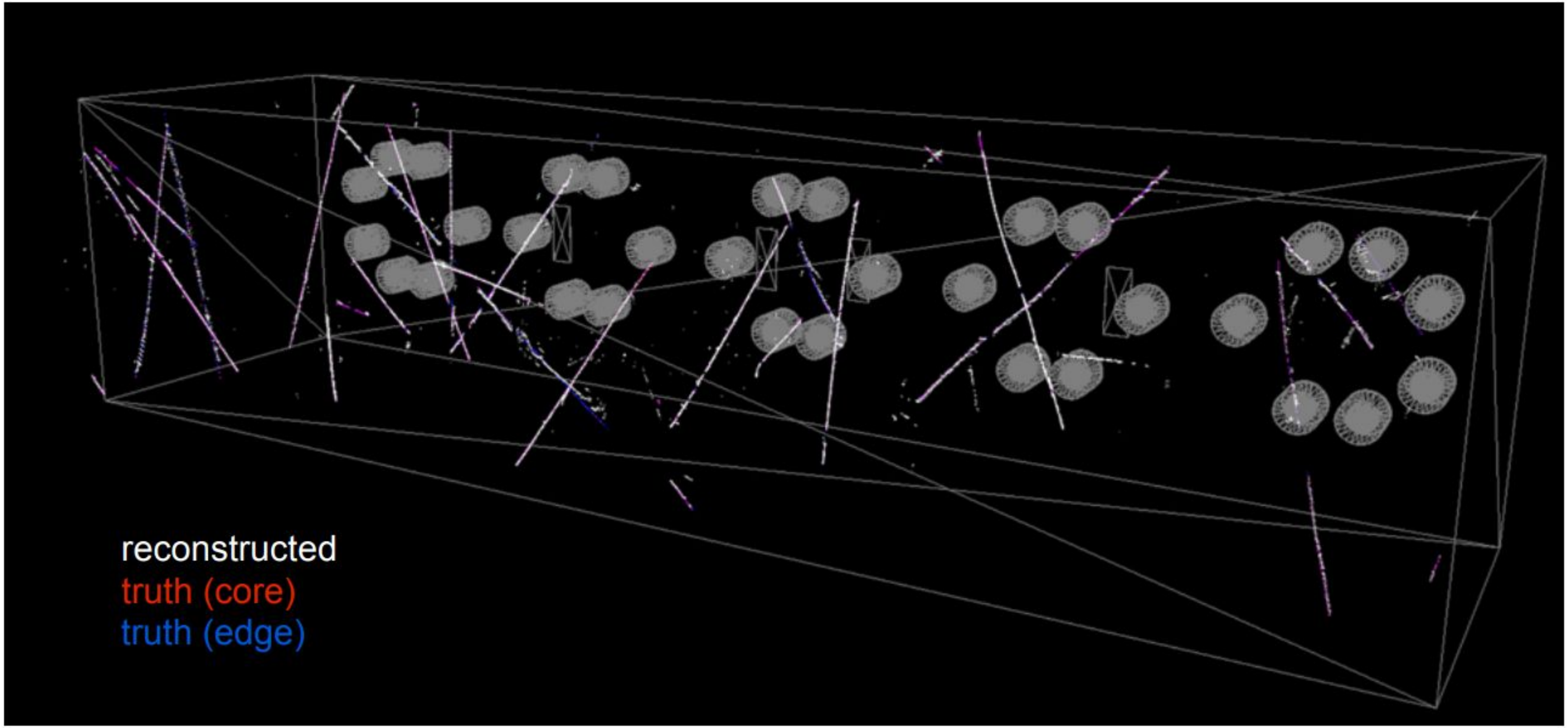

Taking y-plane charge, and moving it to predicted u-plane location

# 3D View



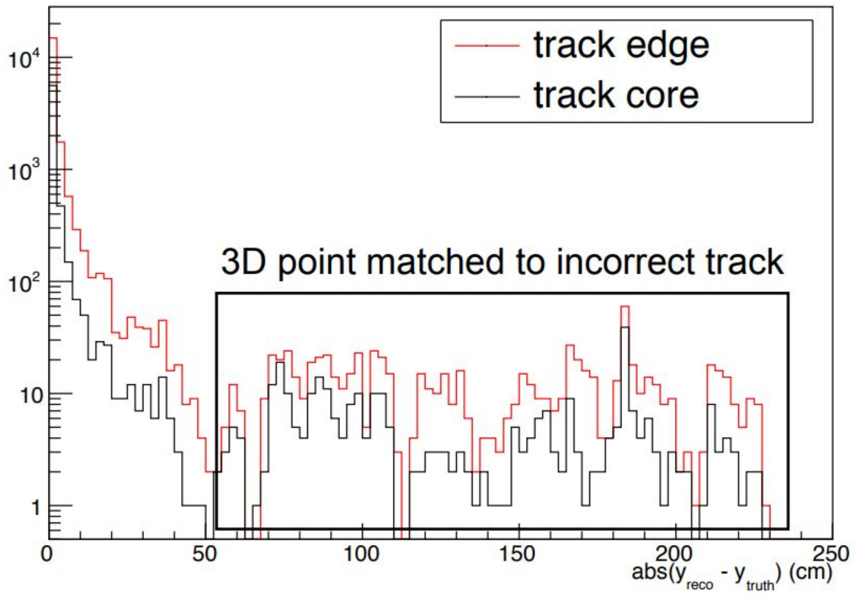

White: reco larflow points
Red lines: MC truth tracks (no space-charge) which 1) cross TPC + 2) visible in image
Blue lines: MC truth tracks (no space-charge) which 1) cross TPC
Green line: MC truth track for neutrino

# 3D View



# Performance Metric (for MC)

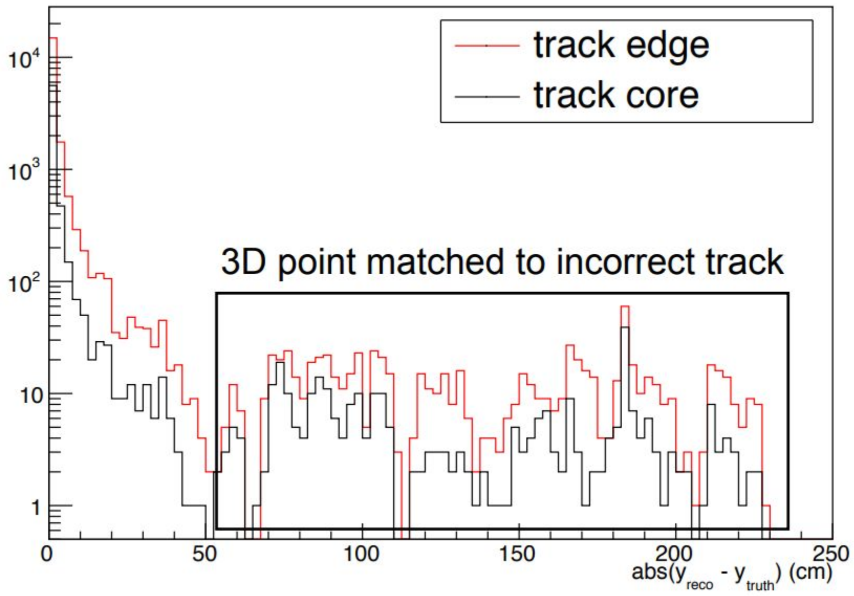
Absolute distance in y (cm) between reco and truth




Within 10cm for 92% of hits
Within 50cm for 95% of hits
If flow prediction (U or V wire) is wrong, we shift to incorrect y

# Performance Metric (for MC)

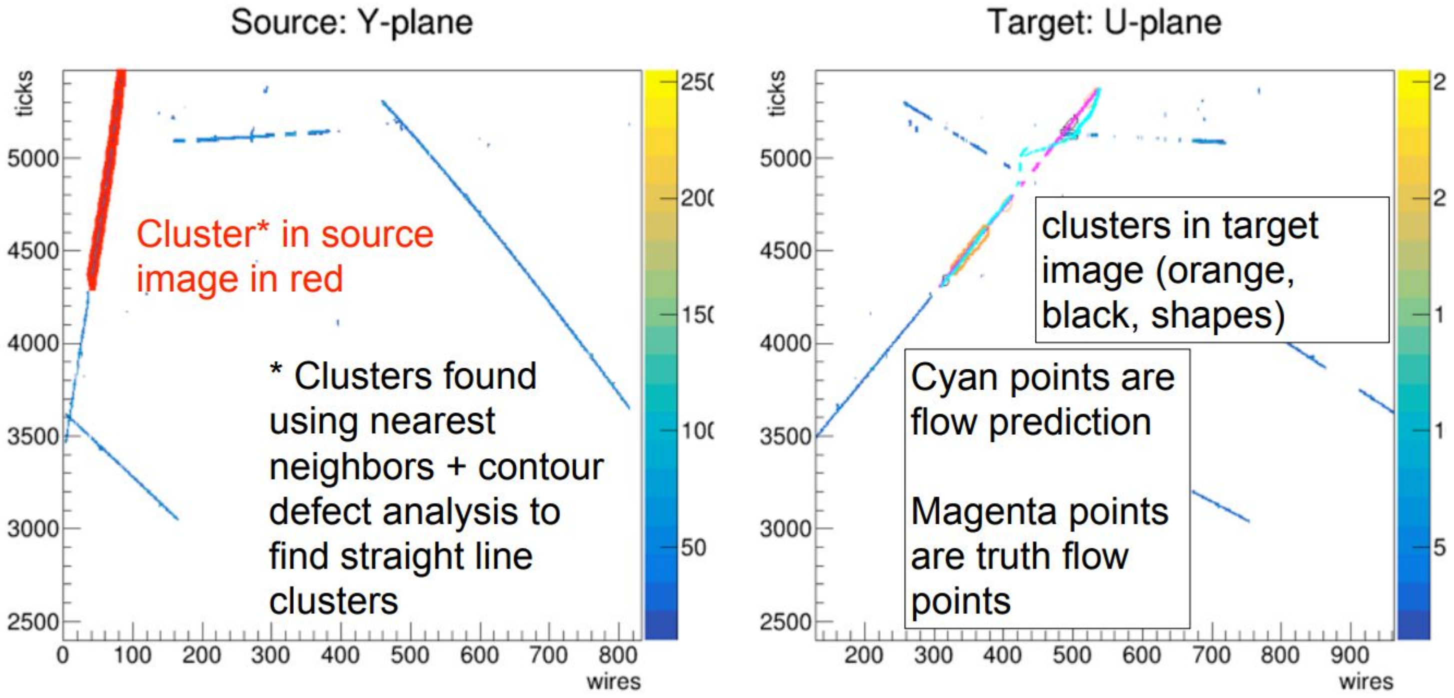
Absolute distance in y (cm) between reco and truth




Have plans to use cosmic muon data to evaluate similar metrics

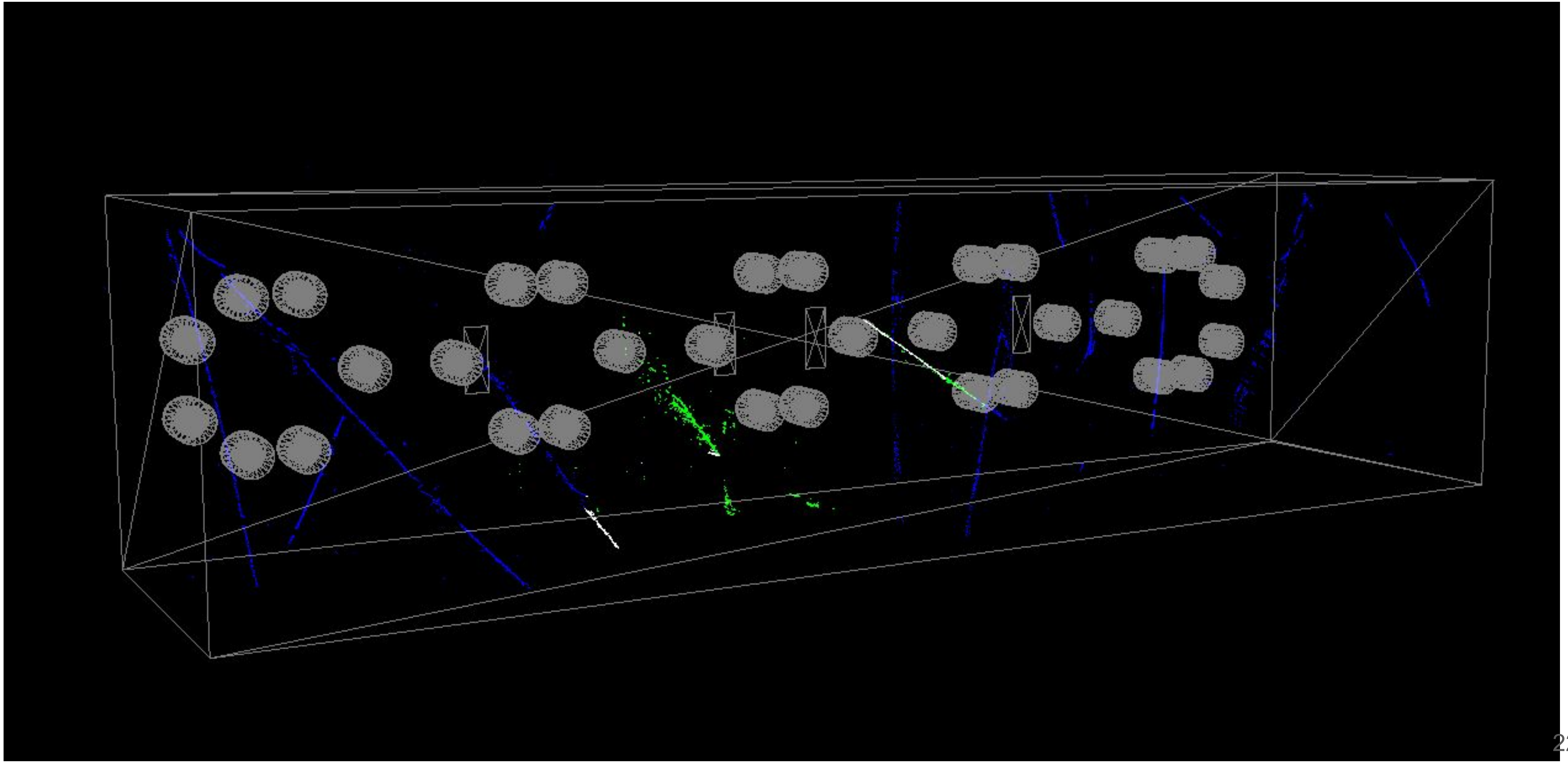
Within 10cm for 92% of hits
Within 50cm for 95% of hits
If flow prediction (U or V wire) is wrong, we shift to incorrect y

# Post-processing

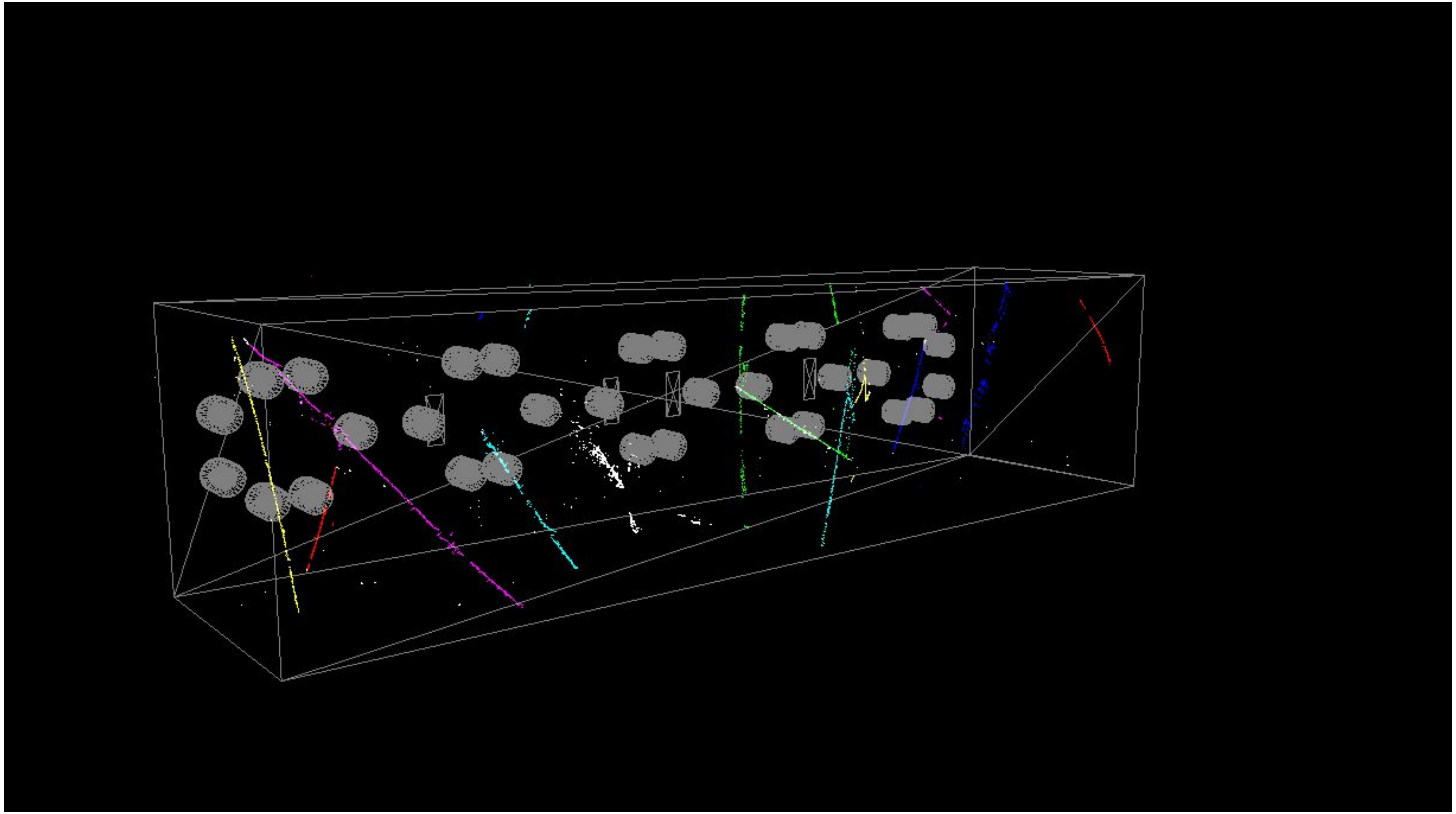



1. For every charge above ADC threshold in Y network predicts pair U (V) wire
2. 2D contours formed on source (Y) and target (U,V) images
3. Points in source clusters matched to target clusters, match quality criteria applied

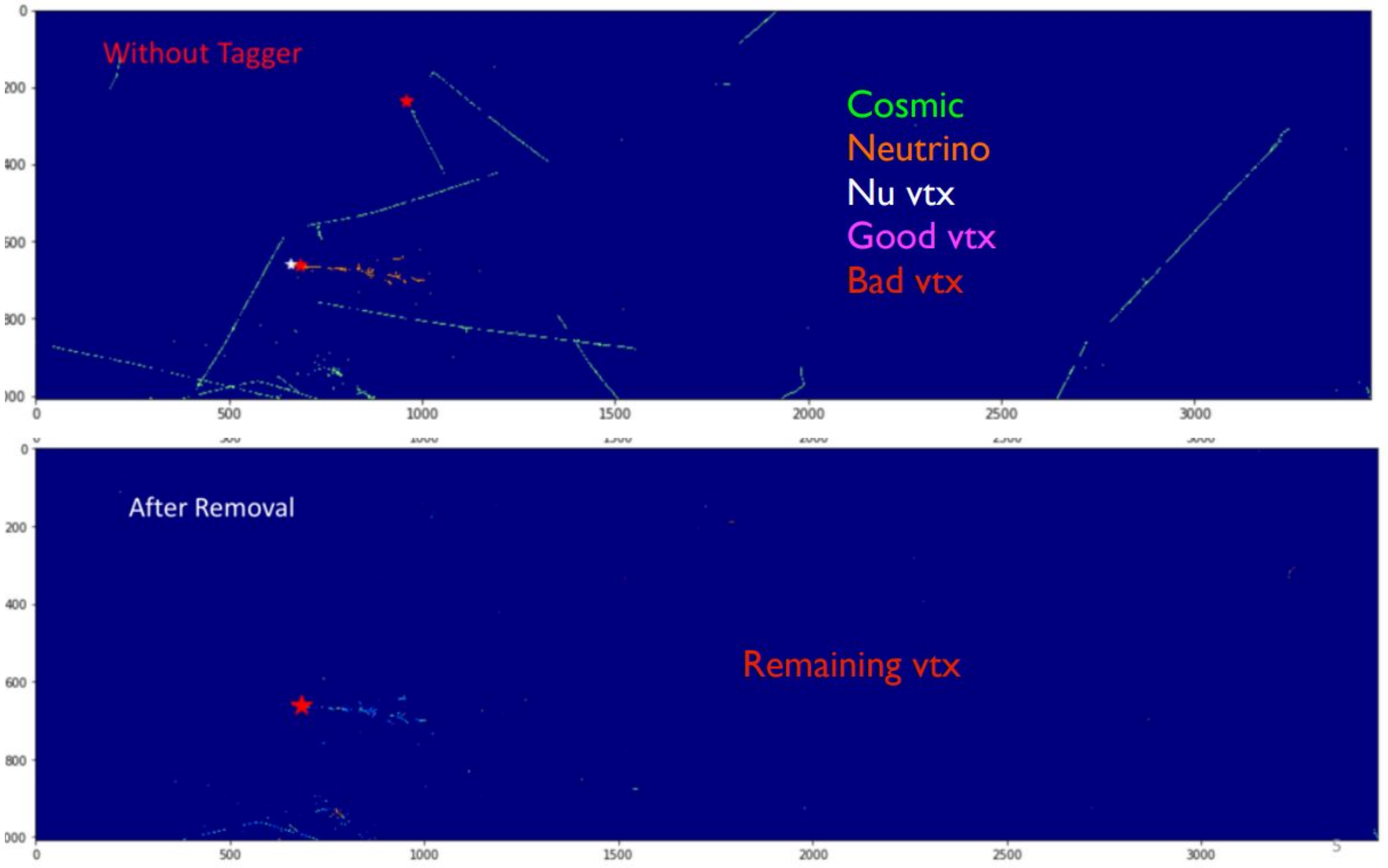
# W/ track/shower topology labeling



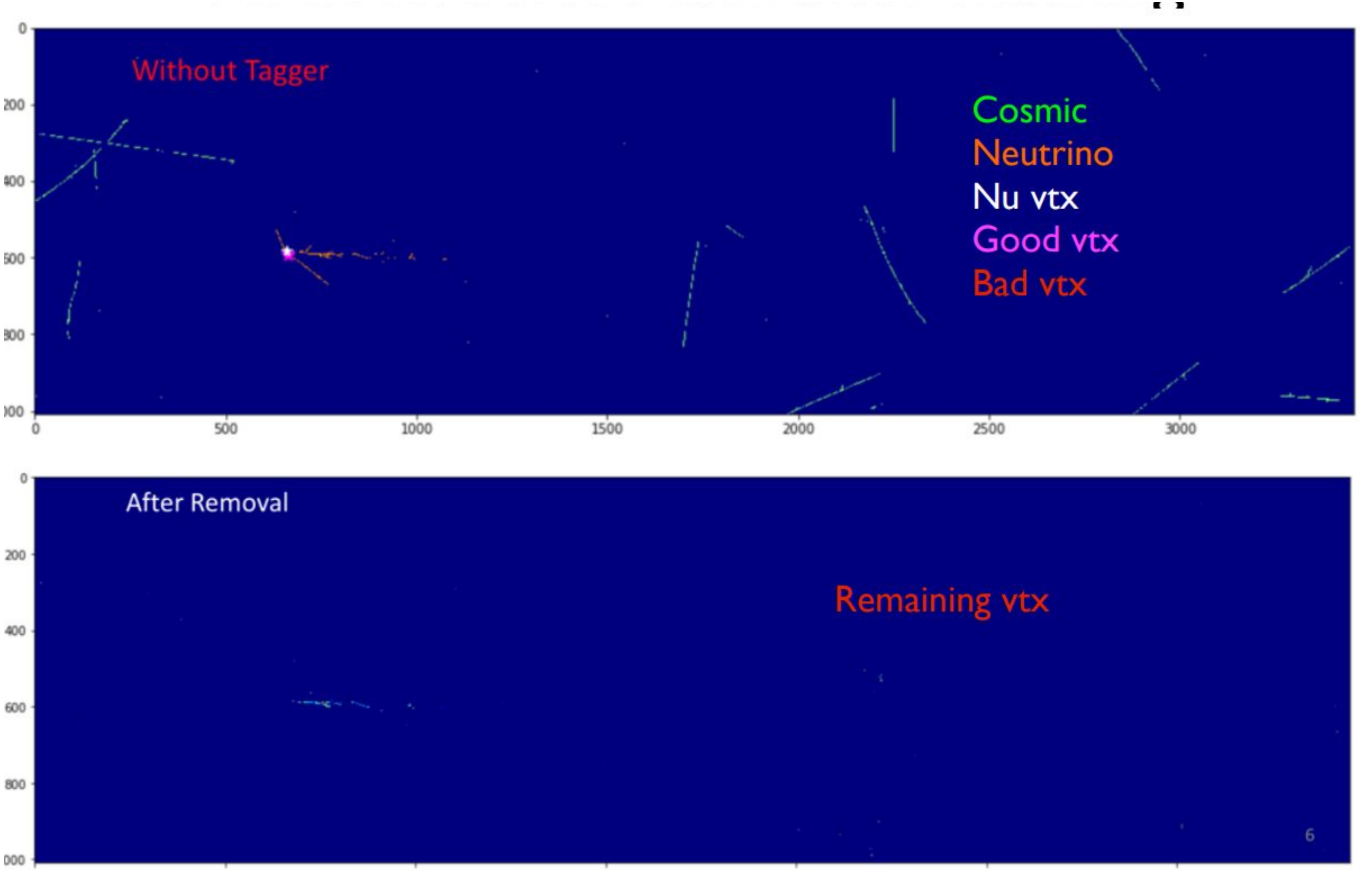
# w/ DL-based clustering (Mask-RCNN)



# First use: false-vertex rejection



# First use: false-vertex rejection




*Good vtx: <5 cm from true neutrino vtx

# Deployment

- With sparse-operations plan is to deploy on single CPU nodes (on FermiGrid)
- First tests on laptop
  - Dual flow prediction: 0.1 seconds (ave use of about 1.2 cores, 1.1 GB)
- With need to split image and merge output of net, *non-network processing* is now the bottleneck (not considering IO)

# Next Steps/Open Questions

- Network optimization
  - Depth and width not explored -- memory limited when using dense conv. operations. With sparse operations can explore more with available hardware
- Visibility prediction
  - Currently off (again due to memory constraints). Now can train.
- Loss improvements
  - 3D consistency: Have redundant predictions. Flow from one plane to the two planes should produce the same 3D position.  Can penalize based on difference in distance. Will it help?
  - Instead of regression, use classifier type losses with each output class being some flow -- seen examples where these are better at learning multi-modal distributions
- Up-weight, up-sample “difficult” examples -- where must decide between two possible regions, areas of large intersection, images with many EM showers

# Summary

- Using CNNs to provide low-level 3D hits as a foundation for point-cloud-based reconstruction techniques (“traditional” and ML-based)
- Good enough accuracies for early uses in cosmic rejection
- Use of sparse submanifold convolutions key in being able to train with large batch sizes and deploy in reasonable time -- opens up exploration of bigger network

Showing work of group members:

Ralitsa Sharanova
(post-doc)

Katie Mason
(grad)

Joshua Mills
(grad)